15/01/2024
Monday

* **Longest increasing subsequence – Important pattern**

Given an integer array nums, return the length of the longest strictly increasing subsequence.

I/P- nums = [10, 9, 2, 5, 3, 7, 101, 18]

O/P- 4

Explanation- The LIS is [2, 3, 7, 101] and its length is 4. There could be many more like above with length 4 in array nums but it doesn't matter what it contains. What matter is that, it should have maximum length.

Approach-

let we have two pointers curr and prev for comparison. Initially, curr is at 0th index and prev is at some invalid index. let it be -1.

for every element in array nums, we have two choices; either include it or exclude it. When →

- Include = (curr = curr+1) and (prev = curr)

prev = curr isliye kiya because last included element nums[curr] hoga. Ab nums array ke next element ko include krna hai ya nhi vo depend krega last included element pr or yha

pr, we are using prev to indicate that this is the last included element. That's why, we are doing prev = curr.

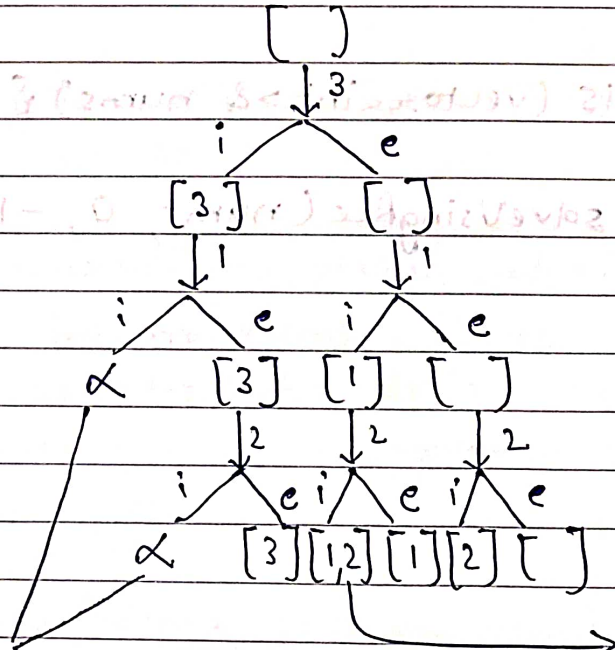
- Exclude = (curr = curr+1) and prev remains as it is.

As yha pe nums[curr] ko include hi nhi kiya to last included element change nhi hua hoga. That's why prev change nhi hua.

→ Here, we are using curr for traversing the nums array. So curr increment hoga hi either we include the current element or not..

Let nums array be —

0	1	2
3	1	2



This is the LIS with length equal 2.

Ye calls isliye ni gyi because nums[curr] last included element i.e. nums[prev] se smaller tha

__/__/____

- Code using recursion -

```
class Solution{
public:

int solveUsingRec (vector<int>&nums, int curr, int prev){

    // base case
    if (curr >= nums.size()) return 0;

    int include = 0;
    if (prev == -1 || nums[curr] > nums[prev]){
      include = 1+ solveUsingRec (nums, curr+1, curr);
    }
    int exclude= 0+ solveUsingRec (nums, curr+1, prev);

    return max (include, exclude);
}

int lengthOfLIS (vector<int>& nums) {

    return solveUsingRec (nums, 0, -1);
}
};
```

__/__/____

- **Code using memoization -**

```
class Solution{
public:

int solveUsingMemo(vector<int>& nums, int curr,
                   int prev, vector<vector<int>>&dp)
{
   if (curr >= nums.size()) return 0;

   if(dp[curr][prev+1] != -1){
      return dp[curr][prev+1];
   }
   int include = 0;
   if(prev == -1 || nums[curr] > nums[prev]){
     include = 1 + solveUsingMemo(nums, curr+1, curr, dp);
   }
   int exclude = 0 + solveUsingMemo(nums, curr+1, prev, dp);

   return dp[curr][prev+1] = max(include, exclude);
}

int lengthOfLIS(vector<int>& nums){
      int n= nums.size();
  vector<vector<int>> dp(n+1, vector<int>(n+1, -1));
      return solveUsingMemo(nums, 0, -1, dp);
  }
};
```

Note -> Yha pe humne index shifting ki he. Initially prev = -1 he or Jb hum dp[curr][prev] krenge to ye runtime error dega. So, dp[curr][prev+1] krenge. Is se ye hoga ki prev = -1 ka ans index 0 pr store hoga and prev=0 ka index 1 pr and so on index shift ho jayenge.

- Code using Tabulation -

```
int solveUsingTabu (vector<int>&nums){

    int n=nums.size();
    vector<vector<int>>dp(n+1, vector<int>(n+1, 0));

    for(int curr=n-1; curr>=0; curr--){
        for(int prev=curr-1; prev>=-1; prev--){
            int include=0;
            if(prev==-1 || nums[curr]>nums[prev]){
                include= 1+ dp[curr+1][curr+1];
            }
            int exclude= 0 + dp[curr+1][prev+1];

            dp[curr][prev+1] = max(include, exclude);
        }
    }
    return dp[0][0];
}
```

• Space optimisation —

```
int solveUsingSO( vector<int>& nums){

    int n = nums.size();

    vector<int> nextRow (n+1, 0);
    vector<int> currRow(n+1, 0);

   for(int curr=n-1; curr>=0; curr--){
    for(int prev=curr-1; prev>=-1; prev--){
     int include=0;
     if(prev==-1 || nums[curr] >nums[prev]){
       include = 1+ nextRow[curr+1];
      }
     int exclude= 0+ nextRow[prev+1];

     currRow[prev+1] = max(include, exclude);
   }
    //shifting
    nextRow = currRow;
}

 return nextRow[0];
}
```

__/__/____

- Using Binary Search for finding length of longest increasing subsequence.

Approach-

(1) Make a vector to store the subsequence.

(2) Store the first element of nums array in that vector.

(3) for the rest of nums array -

- include = if (nums[i] > ans.back())
- overwrite = if (nums[i] < ans.back())

Replace or overwrite the lower_bound of nums[i] with nums[i] in that vector.

(4) Now, the size of vector is the length of LIS.

⟶ let nums be nums = [3, 10, 2, 1, 20]

ans = [3]
10 > 3 ✓
ans = [3, 10]
2 > 10 ✗ (Replace lower_bound of 2 with 2)
ans = [2, 10]
1 > 10 ✗ (Replace lower_bound of 1 with 1)
ans = [1, 10]
20 > 10 ✓
ans = [1, 10, 20]

= size of ans vector is length of LIS i.e 3.

Code-

```
class Solution{
public:

int dpWithBS (vector<int>& nums){
    vector<int>ans;
    ans.push_back(nums[0]);

    for(int i=1; i<nums.size(); i++){
      if(nums[i] > ans.back()){
        //include
        ans.push_back(nums[i]);
      }
      else{
        //overwrite
      int index = lower_bound(ans.begin(), ans.end(),
                              nums[i])-ans.begin();
        ans[index]= nums[i];
      }
    }

    return ans.size();
  }

int lengthOfLIS (vector<int>&nums){

      return dpWithBS(nums);
  }
};
```

Note- lower_bound( ) iterator return krega or ans.begin() is also an iterator. Inko subtract krenge to distance mil jayega i.e. index mil jayega nums[i] ke lower_bound ka.

_/_/_

* **Maximum height by stacking cuboids –**

Given n cuboids where dimension of ith cuboid is cuboids[i] = [w_i, l_i, h_i].
Cuboid i can be placed on cuboid j if $w_i <= w_j$, $l_i <= l_j$ and $h_i <= h_j$. Dimensions of cuboid can be rearranged.
Return maximum height of stacked cuboids.

I/P – cuboids = [[38,25,45],[76,35,3]]

O/P – 76

Approach –

(1) Sort individual cuboid in cuboids array in increasing order. Is se ye hoga, hr cuboid ki maximum dimension last column mei aa jayegi to usko hum height consider kr lenge maximum height bnane ke liye.

(2) Now, sort the cuboids array. Is se ye hoga maximum width ya length wala cuboid sbse last row me aa jayega and height maximize kr paayenge stack ki.

(3) Ab jo cuboids array bna he uspe LIS lgado.

Code -

```
class Solution{
public: // curr[0] >= prev[0] ko check krne ki need
// ni he kyuki vo already sorted order me maintain he
bool check(vector<int>&curr, vector<int>& prev){
    if(curr[1] >= prev[1] && curr[2] >= prev[2]){
        return true;
    }
    else return false;
}

int solve(vector<vector<int>>&nums){
    int n=nums.size();
    vector<int> nextRow(n+1,0);
    vector<int> currRow(n+1,0);

    for(int curr=n-1; curr>=0; curr--){
     for(int prev=curr-1; prev>=-1; prev--){
       int include=0;
       if(prev==-1 || check(nums[curr],nums[prev]))
       { int height=nums[curr][2];
         include= height+nextRow[curr+1];
       }
       int exclude= 0+nextRow[prev+1];

       currRow[prev+1]= max(include,exclude);
     }
     //shifting
     nextRow= currRow;
    }
  return nextRow[0];
}
```

__/__/____

```
int maxHeight (vector<vector<int>>&cuboids){
    for(auto &i : cuboids){
        sort(i.begin(), i.end());
    }

    sort(cuboids.begin(), cuboids.end());

    return solve(cuboids);
  }
};
```

__/__/__

* **Russian Doll Envelopes –**

Given 2D array envelopes where envelopes[i] = [w_i, h_i]. One envelope can fit into another if both width & height of one envelope are greater than the other envelope's width & height.
Return maximum no. of envelopes that can be russian doll. Envelope can't be rotated dimensionally.

I/P– envelopes = [[5,4], [6,4], [6,7], [2,3]]

O/P– 3

Envelopes that can be russian doll –
(2,3) ⟶ (5,4) ⟶ (6,7)

Approach–

(1) Sort the array envelopes in increasing order considering width.

(2) If two widths are equal $w_i = w_j$, sort them in decreasing order considering height.
Now, the question arises – jb width equal ho to height ko decreasing order me sort krdo. Why not in increasing order? Because –

Let envelopes = [[3,5], [4,7], [3,6], [1,3], [2,4]]

⟶ Ab isko agr sort krde in increasing order of width and when $w_i = w_j$, then increasing order of height.

Now, envelopes = [[1,3],[2,4],[3,5],[3,6],[4,7]]

Ab isme agr heights of envelopes pr LIS lgaye i.e. for → 3, 4, 5, 6, 7

The answer would be 5 i.e. all the envelopes can be russian doll according to it.
But we can't put the envelope [3,5] into [3,6] because of having same width.
So, the answer should be 4.

→ Now, if envelopes array ko sort kre in increasing order of width and when $w_i = w_j$, then decreasing order of height.

envelopes = [[1,3],[2,4],[3,6],[3,5],[4,7]]

Ab isme agr height of envelopes pr LIS lgaye i.e. for → 3, 4, 6, 5, 7

The answer would be 4 because LIS me 6 ke baad 5 ko consider nhi krenge.
That's why sort ese krenge.

⇒ Ye Longest increasing subsequence ki tabulation wali approach se nhi hoga because of larger constraints isliye isme binary search wala logic lgayenge.

Code-

```
class Solution {
public:

static bool (vector<int>&a, vector<int>& b){
    if (a[0] == b[0]) return a[1] > b[1];
    return a[0] < b[0];
}

int solve (vector<vector<int >>& nums) {
    sort (nums.begin(), nums.end(), comp);

    vector <int> ans;
    ans.push_back(nums[0][1]);

    for (int i=1; i<nums.size(); i++) {
        if (nums[i][1] > ans.back()){
            //include
            ans.push_back (nums[i][1]);
        }
        else {
            //overwrite
            int index= lower_bound (ans.begin(),
                ans.end(), nums[i][1])-ans.begin
            ans[index]= nums[i][1];
        }
    }
    return ans.size();
}
int maxEnvelopes (vector<vector<int>>&envelopes){
    return solve (envelopes);
  }
};
```